Docket No. AUS920010666US1

METHOD, SYSTEM, AND PRODUCT FOR STORAGE OF ATTRIBUTE DATA IN AN OBJECT ORIENTED ENVIRONMENT

CROSS REFERENCE TO RELATED APPLICATIONS

The present invention is related to the following applications entitled: "Method And Apparatus In A Data Processing System For Refreshing Multiple Data Models In An Application", U.S. Application Serial Number 09/429,212, Attorney Docket Number AUS990339US2, filed on October 28, 1999; "Method and Apparatus in a Data Processing System for Systematically Separating Application Graphical User Interface Component Placement From Component Sequencing And Component Creation", U.S. Application Serial Number 09/429,522, Attorney Docket Number AUS990339US3, filed on October 28, 1999; "Method and Apparatus in a Data Processing System For Systematically Separating Application Components From Legacy System Services", U.S. Application Serial Number 09/430,355, Attorney Docket Number AUS990339US4, filed on October 28, 1999; "Method and Apparatus in a Data Processing System For The Issuance And Delivery Of Lightweight Requests To Concurrent And Multiple Service Providers", U.S. Application Serial Number 09/430,814, Attorney Docket Number AUS990339US5, filed on October 29, 1999; "Method and Apparatus in a Data Processing System For The Controlling And Sequencing Of Graphical User Interface Components And Mediating Access To System Services For Those Components", U.S. Application Serial Number 09/430,324, Attorney Docket Number AUS990339US6, filed on October 29, 1999; "Method and Apparatus In A Data Processing System For Defining And Composing

Type-Based (Multi-Level) Field Validation and Formatting Rules", U.S. Application Serial Number 09/430,824, Attorney Docket Number AUS990339US7, filed on October 29, 1999; "Method and Apparatus in a Data Processing System For The Separation Of Role-Based Permissions Specification From Its Corresponding Implementation Of Its Semantic Behavior", U.S. Application Serial Number 09/430,861, Attorney Docket Number AUS990339US8, filed on October 29, 1999; "Method and Apparatus in a Data Processing System For Generating Alternate Views of Client Applications", U.S. Application Serial Number 09/430,823, Attorney Docket Number AUS990339US9, filed on October 29, 1999; "Method and Apparatus in a Data Processing System Of Creating Highly Reusable Application Function Sequencers Using Graph-Based Partitioning", U.S. Application Serial Number 09/429,528, Attorney Docket Number AUS990339US10, filed on October 28, 1999; "Method and Apparatus in a Data Processing System For Specifying The Sequencing And Mediation Of Application Components", U.S. Application Serial Number 09/431,429, Attorney Docket Number AUS990339US11, filed on October 29, 1999; "Method and Apparatus in a Data Processing System For Systematically Serializing Complex Data Structures", U.S. Application Serial Number 09/429,593, Attorney Docket Number AUS990339US13, filed on October 28, 1999; "Method and Apparatus in a Data Processing System For Processing Events In An Object Oriented Environment", U.S. Application Serial Number 09/429,592, Attorney Docket Number AUS990339US14, filed on October 28, 1999; all of which are assigned to the same assignee.

BACKGROUND OF THE INVENTION

1. Technical Field:

The present invention relates generally to an improved distributed data processing system, and, in particular to an improved system, method, and computer program product within an object oriented environment. Still more particularly, the present invention relates to a system, method, and computer program product for storage of attribute data in an object oriented environment in such a manner as to reduce the logic to store attributes and the amount of storage space for those attributes.

2. Description of Related Art:

Object-oriented languages have been employed in creating applications. One of the big advantages of object-oriented languages is the ability to reuse portions of the software code. Reuse of the code is achieved through inheritance. In many cases, inheritance can make development of new classes very simple because the logic of a parent class need not be reinvented for each child of that parent class.

This advantage does not normally apply, however, to functions that act on the attributes of a class. These functions must be rewritten for each subclass so that the attributes of each subclass are acted upon. One such example is when the attributes of an object must be converted to another representation, such as XML data, or a byte stream.

Figure 1 depicts an object definition having attributes defined within classes and having attribute

data stored within the class in which the attribute is defined in accordance with the prior art. Object **10** includes an ID class **12**, a person class **14**, and a customer class **16**. Person class **14** is a subclass of ID class **12**. And, customer class **16** is a subclass of person class **14**.

An object identifier (ID) attribute is defined within ID class **12**. A first name attribute and a last name attribute are defined within person class **14**. A customer number attribute and a last visit date attribute are defined within customer class **16**. The object identifier data will be stored in ID class **12**. The first name data and last name data will both be stored in the person class **14**. The customer number data and last visit date data will both be stored in the customer class **16**. Thus, the attribute data is stored within the class in which the associated attribute is defined.

Figure 2 illustrates a particular instance of an object **20**, having an object definition defined as depicted by **Figure 1**, in accordance with the prior art. An object identifier, "12345", is stored within ID class **22**. A first name, "John", and a last name, "Doe", are both stored within person class **24**. A customer number, "3413523", and a last visit date, "3/12/2001", are both stored within customer class **26**.

Methods, or logic, that need to act upon all attribute data of object **10** must be written for each class for which one or more attributes are defined. As depicted by **Figure 2**, a "read object" method and a "write object" method are both defined within each class, i.e. ID class **22**, person class **24**, and customer class **26**. The

"read object" method will read all of the attribute data for only the class within which the "read object" method is defined. Similarly, the "write object" method will write all of the attribute data for only the class within which the "write object" method is defined. In order to act upon all attribute data of object **20**, these methods must be defined for each class of object **20**.

The read and write object methods defined within ID class **22** will read and write only the object ID data, "12345". The read and write object methods defined within person class **24** will read and write only the first name and last name data, "John" and "Doe". The read and write object methods defined within customer class **26** will read and write only the customer number and last visit date data, "3413523" and "3/12/2001".

Thus, as depicted by **Figures 1** and **2**, the code, or logic, must be written for each class of an object in order to act upon all of the attributes of the object.

Therefore, it would be advantageous to have an improved method, system, and product for storage of attribute data in an object oriented environment in such a manner as to reduce attribute logic contained in the "readObject" and "writeObject" methods.

SUMMARY OF THE INVENTION

A method, system, and computer program product are described for storing attribute data in an object oriented environment. A base class and a subclass are defined within an object. The object is defined within the object oriented environment. An attribute is defined within the subclass. The attribute data defined for the subclass is stored only within the base class, and is not stored within the subclass.

The above as well as additional objectives, features, and advantages of the present invention will become apparent in the following detailed written description.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 depicts an object definition having attributes defined within classes and having attribute data stored within the class in which the attribute is defined in accordance with the prior art;

Figure 2 illustrates an instance of an object, having an object definition defined as depicted by **Figure 1**, in accordance with the prior art;

Figure 3 depicts a pictorial representation of a distributed data processing system in which the present invention may be implemented;

Figure 4 is a block diagram depicting a data processing system that may be implemented as a server in accordance with a preferred embodiment of the present invention;

Figure 5 is a block diagram illustrating a data processing system in which the present invention may be implemented;

Figure 6 depicts an object definition having attributes defined within classes but having all attribute data for these attributes stored in a base superclass in accordance with the present invention;

Figure 7 illustrates an instance of an object where all attribute data is stored only in a base class, where the object has an object definition defined as depicted by **Figure 6**, in accordance with the present invention;

Figure 8 is a diagram of a serializer system in accordance with a preferred embodiment of the present invention; and

Figure 9 is a diagram of a deserializer system in accordance with a preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention provides for storage of all attribute data of an object within a specialized base class. The base class is the superclass of the object.

One or more attributes are defined within subclasses of the object. A separate index is defined for and associated with each attribute. The attribute data for an attribute, along with the index defined for that attribute, are both stored in a storage attribute within the base class.

In this manner, no attribute data is stored in the subclasses of the object. All of the attribute data is stored in a storage attribute defined within the base class. By storing all of the attribute data in a base, superclass, the base class has knowledge of and access to all attribute data.

With reference now to the figures, **Figure 3** depicts a pictorial representation of a distributed data processing system in which the present invention may be implemented. Distributed data processing system **300** is a network of computers in which the present invention may be implemented. Distributed data processing system **300** contains a network **302**, which is the medium used to provide communications links between various devices and computers connected together within distributed data processing system **300**. Network **302** may include permanent connections, such as wire or fiber optic cables, or temporary connections made through telephone connections.

In the depicted example, a server **304** is connected to network **302** along with storage unit **306**. In addition, clients **308**, **310**, and **312** also are connected to a network

302. These clients **308**, **310**, and **312** may be, for example, personal computers or network computers. For purposes of this application, a network computer is any computer, coupled to a network, which receives a program or other application from another computer coupled to the network. In the depicted example, server **304** provides data, such as boot files, operating system images, and applications to clients **308-312**. Clients **308**, **310**, and **312** are clients to server **304**. Distributed data processing system **300** may include additional servers, clients, and other devices not shown.

In the depicted example, distributed data processing system **300** is the Internet with network **302** representing a worldwide collection of networks and gateways that use the TCP/IP suite of protocols to communicate with one another. At the heart of the Internet is a backbone of high-speed data communication lines between major nodes or host computers, consisting of thousands of commercial, government, educational and other computer systems that route data and messages. Of course, distributed data processing system **300** also may be implemented as a number of different types of networks, such as for example, an intranet, a local area network (LAN), or a wide area network (WAN). **Figure 3** is intended as an example, and not as an architectural limitation for the present invention.

Referring to **Figure 4**, a block diagram depicts a data processing system that may be implemented as a server, such as server **304** in **Figure 3**, in accordance with a preferred embodiment of the present invention. Data processing system **400** may be a symmetric multiprocessor (SMP) system including a plurality of

processors **402** and **404** connected to system bus **406**. Alternatively, a single processor system may be employed. Also connected to system bus **406** is memory controller/cache **408**, which provides an interface to local memory **409**. I/O bus bridge **410** is connected to system bus **406** and provides an interface to I/O bus **412**. Memory controller/cache **408** and I/O bus bridge **410** may be integrated as depicted.

Peripheral component interconnect (PCI) bus bridge **414** connected to I/O bus **412** provides an interface to PCI local bus **416**. A number of modems may be connected to PCI bus **416**. Typical PCI bus implementations will support four PCI expansion slots or add-in connectors. Communications links to network computers **308-312** in **Figure 3** may be provided through modem **418** and network adapter **420** connected to PCI local bus **416** through add-in boards.

Additional PCI bus bridges **422** and **424** provide interfaces for additional PCI buses **426** and **428**, from which additional modems or network adapters may be supported. In this manner, server **400** allows connections to multiple network computers. A memory-mapped graphics adapter **430** and hard disk **432** may also be connected to I/O bus **412** as depicted, either directly or indirectly.

Those of ordinary skill in the art will appreciate that the hardware depicted in **Figure 4** may vary. For example, other peripheral devices, such as optical disk drives and the like, also may be used in addition to or in place of the hardware depicted. The depicted example is not meant to imply architectural limitations with respect to the present invention.

The data processing system depicted in **Figure 4** may be, for example, an IBM RISC/System 6000 system, a product of International Business Machines Corporation in Armonk, New York, running the Advanced Interactive Executive (AIX) operating system.

With reference now to **Figure 5**, a block diagram illustrates a data processing system in which the present invention may be implemented. Data processing system **500** is an example of a client computer. Data processing system **500** employs a peripheral component interconnect (PCI) local bus architecture. Although the depicted example employs a PCI bus, other bus architectures such as Micro Channel or Industry Standard Architecture (ISA) may be used. Processor **502** and main memory **504** are connected to PCI local bus **506** through PCI bridge **508**. PCI bridge **508** also may include an integrated memory controller and cache memory for processor **502**. Additional connections to PCI local bus **506** may be made through direct component interconnection or through add-in boards. In the depicted example, local area network (LAN) adapter **510**, Small Computer System Interface host bus adapter **512**, and expansion bus interface **514** are connected to PCI local bus **506** by direct component connection. In contrast, audio adapter **516**, graphics adapter **518**, and audio/video adapter **519** are connected to PCI local bus **506** by add-in boards inserted into expansion slots. Expansion bus interface **514** provides a connection for a keyboard and mouse adapter **520**, modem **522**, and additional memory **524**. Small computer system interface (SCSI) host bus adapter **512** provides a connection for hard disk drive **526**, tape drive

528, and CD-ROM drive 530. Typical PCI local bus implementations will support three or four PCI expansion slots or add-in connectors.

An operating system runs on processor 502 and is used to coordinate and provide control of various components within data processing system 500 in **Figure 5**. The operating system may be a commercially available operating system such as OS/2, which is available from International Business Machines Corporation. "OS/2" is a trademark of International Business Machines Corporation. An object oriented programming system such as Java may run in conjunction with the operating system and provides calls to the operating system from Java programs or applications executing on data processing system 500. "Java" is a trademark of Sun Microsystems, Inc. Instructions for the operating system, the object-oriented operating system, and applications or programs are located on storage devices, such as hard disk drive 526, and may be loaded into main memory 504 for execution by processor 502.

Those of ordinary skill in the art will appreciate that the hardware in **Figure 5** may vary depending on the implementation. Other internal hardware or peripheral devices, such as flash ROM (or equivalent nonvolatile memory) or optical disk drives and the like, may be used in addition to or in place of the hardware depicted in **Figure 5**. Also, the processes of the present invention may be applied to a multiprocessor data processing system.

For example, data processing system 500, if optionally configured as a network computer, may not include SCSI host bus adapter 512, hard disk drive 526,

tape drive **528**, and CD-ROM **530**, as noted by dotted line **532** in **Figure 5** denoting optional inclusion. In that case, the computer, to be properly called a client computer, must include some type of network communication interface, such as LAN adapter **510**, modem **522**, or the like. As another example, data processing system **500** may be a stand-alone system configured to be bootable without relying on some type of network communication interface, whether or not data processing system **500** comprises some type of network communication interface. As a further example, data processing system **500** may be a Personal Digital Assistant (PDA) device, which is configured with ROM and/or flash ROM in order to provide nonvolatile memory for storing operating system files and/or user-generated data.

The depicted example in **Figure 5** and above-described examples are not meant to imply architectural limitations. For example, data processing system **500** also may be a notebook computer or hand held computer in addition to taking the form of a PDA. Data processing system **500** also may be a kiosk or a Web appliance.

Figure 6 depicts an object definition having attributes defined within classes but having all attribute data for these attributes stored only in a base superclass in accordance with the present invention. Object **600** includes a base class **602**, an ID class **604**, a person class **606**, and a customer class **608**. ID class **604** is a subclass of base class **602**. Person class **606** is a subclass of ID class **604**. And, customer class **608** is a subclass of person class **606**.

Base class **602** includes a storage attribute, "Object[] data" which is the Java definition for an array of objects of type "Object". All attribute data of object **600** will be stored in this storage attribute. Programmers with ordinary skill in the art will realize that other data storage structures such as a vector could be used instead.

The number of attributes defined by all of the subclasses of object **600** must be provided to base class **602** when object **600** is created. Base class **602** then sets aside the required array size to accommodate the total number of attributes. When object **600** is created these indexes are predefined and associated with particular attributes. Subclasses may access their attribute data by referencing the appropriate array index.

An object identifier (ID) attribute is defined within ID class **604**. An index of "0" is defined and associated with the object identifier attribute. The object identifier data is not stored in the ID class **604**. The object identifier data is stored in the "Object[] data" attribute at the "0" index.

A first name attribute and a last name attribute are defined within person class **606**. An index of "1" is defined and associated with the first name attribute, and an index of "2" is defined and associated with the last name attribute. The first name data is stored in the "Object[] data" attribute at the "1" index. The last name data is stored in the "Object[] data" attribute at the "2" index.

A customer number attribute and a last visit date attribute are defined within customer class **608**. An index of "3" is defined and associated with the customer

number attribute, and an index of "4" is defined and associated with the last visit date attribute. The customer number data is stored in the "Object[] data" attribute at the "3" index. The last visit date data is stored in the "Object[] data" attribute at the "4" index.

Thus, all of the attribute data for object **600** is stored in the base class within the storage attribute, "Object[] data". None of this attribute data is stored within ID class **604**, person class **606**, or customer class **608**.

In order to prevent errors and allow for the classes to be more easily maintained, a constant should be defined in each class to indicate the last index used by that class. Each subclass should define their indexes relative to the last index used by the super class, indicated by the constant. When an attribute is added to a class, the last index will be updated and all of the subclasses, through the use of the constant, will be automatically updated.

Because all of the attribute data is stored within the base class, methods that need to act upon all attribute data need to be written only for the base class. As depicted by **Figure 6**, a "read object" method and a "write object" method are both defined within base class **602**. The "read object" method will read all of the attribute data for object **600**, and the "write object" method will write all of the attribute data for object **600**. These methods do not need to be defined for each subclass. Thus, these methods are not defined for ID class **604**, person class **606**, or customer class **608**.

Figure 7 illustrates an instance of an object **700**, which is defined as depicted by **Figure 6**, where all

attribute data is stored only in base class **702** in accordance with the present invention. All attribute data of object **700** is stored within the "Object[] data" storage attribute defined in base class **702**. An object identifier, "12345", is stored at index "0". A first name, "John", is stored at index "1". A last name, "Doe", is stored at index "2". A customer number, "3413523", is stored at index "3". And, a last visit date, "3/12/2001", is stored at index "4". None of this attribute data is stored in the class in which the associated attribute is defined.

One particular example of a case where all attribute data defined for an object needs to be acted upon is when the object is serialized or deserialized. When an object is serialized, all of the attribute data of the object is written to a data stream. When an object is deserialized, all of the attribute data of the object is read from a data stream. Using the present invention, this process of writing and reading all of the attribute data of an object may be accomplished by providing read and write methods only for the base object.

Within the write method, all of the attribute data must be written to a data stream. To accomplish this using the present invention, the size of the data array containing all of the attribute data should be written to the data stream first. After the size of the array is written, each of the elements in the array should also be written to the data stream.

When reading the data from the stream, the information must be read in the same order as it was written. In this case, the size of the data array will be read first. This value is used to create a new data

array which will be used to store the attributes. Each of the attributes that were written to the data stream will then be read and put in the data array at the appropriate index.

The value of the present invention is that all of this logic is contained within a single class. To use a different algorithm for reading and writing the attribute values, the change can be made easily in a single place.

The present invention may be utilized in a serialization and deserialization process such as depicted by **Figures 8** and **9**.

The present invention is related to the subject matter of U.S. Patent 6,292,933 issued to Peter C. Bahrs et al. on September 18, 2001, which is hereby incorporated by reference.

It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media such a floppy disc, a hard disk drive, a RAM, and CD-ROMs and transmission-type media such as digital and analog communications links.

The description of the present invention has been presented for purposes of illustration and description, but is not intended to be exhaustive or limited to the invention in the form disclosed. Many modifications and

variations will be apparent to those of ordinary skill in the art. For example, although the depicted architectural pattern is illustrated in a Java programming environment, the architectural pattern of the present invention may be applied to other types of programming environments. For example, VisualBasic, C++ and Smalltalk are other programming environments in which the processes of the present invention may be applied. In addition, the description of the classes along with the variables, constructors, and methods are provided for purposes of illustration only. Classes, variables, constructors, and methods may vary depending on the particular implementation. Illustration of these classes, variables, constructors are for illustrative purposes and not intended to limit the architectural pattern of the present invention. The embodiment was chosen and described in order to best explain the principles of the invention, the practical application, and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated.